Universidade Federal do Paraná

Laboratório de Estatística e Geoinformação - LEG

Gradiente descendente (batch, stochastic e boosting)

Profs.: Eduardo Vargas Ferreira
Walmes Marques Zeviani

Solução de quadrados mínimos

- Seja $\boldsymbol{X} \in \mathrm{M}_{n\times p}(\mathrm{R})$, com $n > p$ e $posto(\boldsymbol{X}) = p$. Dado $\boldsymbol{y} \in \mathrm{R}^n$, definimos o seguinte problema de minimização:

$$\left\| X\hat{\boldsymbol{\beta}} - \boldsymbol{y} \right\|_2^2 = min\left\{ \|X\boldsymbol{\beta} - \boldsymbol{y}\|_2^2 \ ; \ \boldsymbol{\beta} \in \mathrm{R}^p \right\}$$

- Dizemos que o elemento $\hat{\beta}$ é uma solução de quadrados mínimos;

Teorema: Seja $\boldsymbol{X} \in \mathrm{M}_{n\times p}(\mathrm{R})$, com $n > p$ e $posto(\boldsymbol{X}) = p$. Definimos $J : \mathrm{R}^p \to \mathrm{R}$ da seguinte forma:

$$J(\boldsymbol{\beta}) = \langle \boldsymbol{X\beta} - \boldsymbol{y}, \boldsymbol{X\beta} - \boldsymbol{y} \rangle \ ; \ \boldsymbol{\beta} \in \mathrm{R}^p.$$

Então, o **Problema de Minimização**: encontrar $\hat{\beta} \in \mathrm{R}^p$ tal que

$$J(\hat{\boldsymbol{\beta}}) = min\{J(\boldsymbol{\beta}) \ ; \ \boldsymbol{\beta} \in \mathrm{R}^p\}$$

é equivalente ao **Sistema Normal**

$$\boldsymbol{X}^t\boldsymbol{X}\boldsymbol{\beta} = \boldsymbol{X}^t\boldsymbol{y}.$$

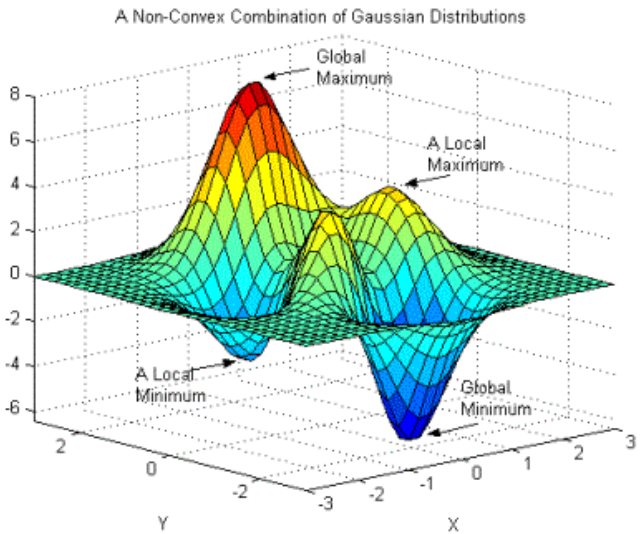
Método do gradiente descendente (GD)

- O **Gradiente descendente** (GD) é um método para encontrar o mínimo de uma função de forma iterativa;

Algoritmo: Escolha um chute inicial, $\beta^{(0)} \in \mathrm{R}^p$, repita:

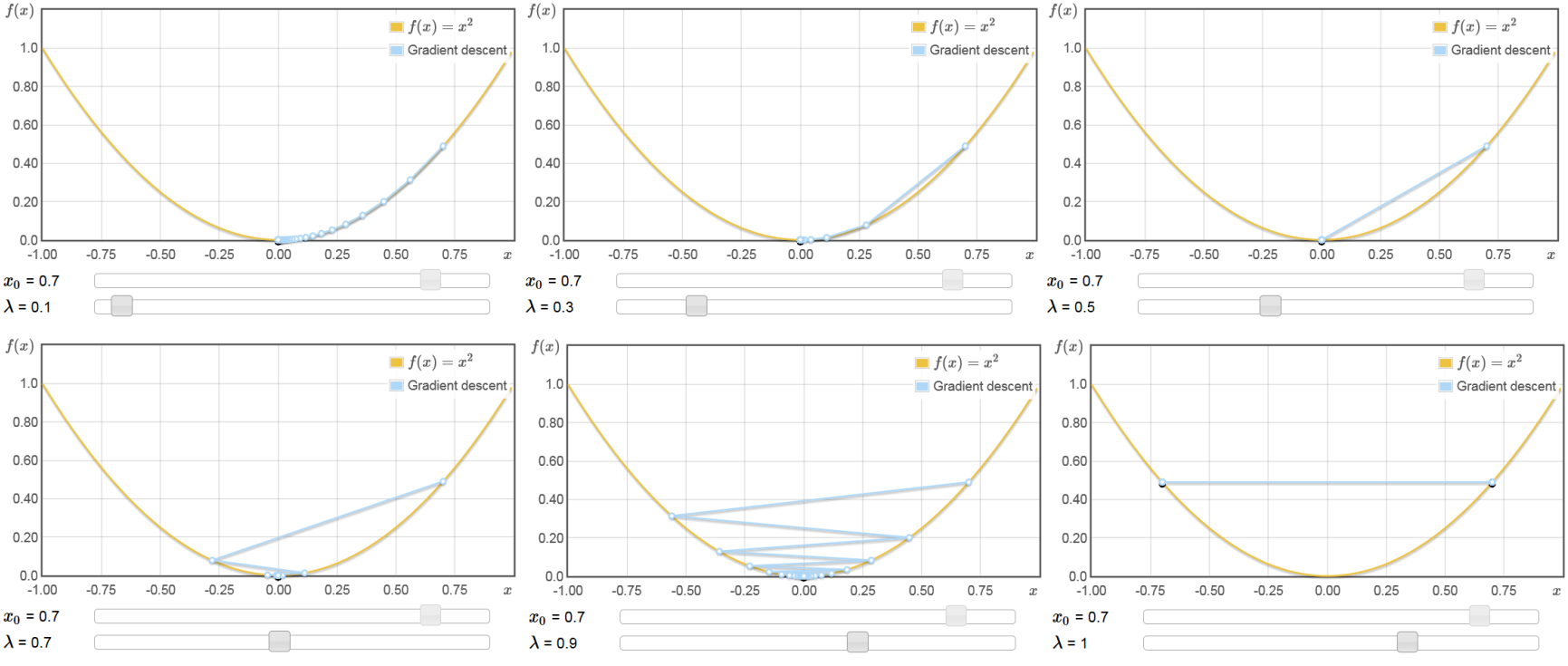
$$\beta^{(k+1)} = \beta^{(k)} - \alpha_k \nabla J(\beta^{(k)}), \ k = 0, 1, \ldots$$

pare quando atingir convergência.



Taxa de aprendizagem α

- Taxa de aprendizagem controla o tamanho do passo em cada iteração;
- Selecionar o valor correto é crítico
  - ⋆ Se tomarmos α pequeno, o método fica lento;
  - ⋆ Se α muito grande, o método diverge.



Prós e contras do GD

✓ Ideia simples e cada iteração é barata;

✓ Garantia de convergência para o mínimo local;

✓ Com vários algoritmos de segunda ordem para acelerar sua convergência;

✓ Muito rápido para matrizes bem condicionadas e problemas fortemente convexos;

χ Frequentemente é lento, pois problemas interessantes não são fortemente convexos ou bem condicionados;

χ Não lida com funções não diferenciáveis (dica: use o método Subgradiente).

χ Utiliza todos os dados de treinamento para estimar os parâmetros. Assim, para grandes bancos de dados, torna-se lento;

- Diante deste último aspecto, por que não em cada iteração selecionar um valor na amostra, e com sua informação executar um passo?

Gradiente descendente estocástico (GDE)

Gradiente descendente estocástico (GDE)

- Como vimos, no Gradiente Descendente utilizamos a **amostra completa** para atualizar os parâmetros (é um processo determinístico);
- Assim, se o tamanho da amostra de treino for grande (na verdade MUITO grande!) o Gradiente Descendente levará muito tempo em cada passo;
- A diferença no **Gradiente Descendente Estocástico** (GDE) está na utilização de somente uma observação em cada iteração;
- Então, cada passo é realizado com uma v.a. de um processo estocástico. Tornando o método mais atrativo;
- Em redes neurais, p.ex., o custo para se fazer **backpropagation** com os dados completos é alto, precisando de abordagens estocásticas como esta.

Gradiente descendente estocástico (GDE)

- Considere o par $(\boldsymbol{x}_i, y_i)$ amostrado do treinamento. A atualização dos parâmetros é dada por

Algoritmo: Escolha um chute inicial, $\boldsymbol{\beta}^{(0)} \in \mathrm{R}^{p+1}$, repita:

$$\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} - \alpha_k \nabla J(\boldsymbol{\beta}^{(k)}; \boldsymbol{x}_i, y_i), \ k = 0, 1, \ldots$$

pare quando atingir convergência.

- No GDE a taxa de aprendizagem, α, é, tipicamente, menor do que o GD (*batch*). Isso ocorre, pois temos uma maior variância nas atualizações;

- Uma escolha de α, que funciona bem na prática, é uma taxa pequena o suficiente que dê uma convergência estável nas iterações iniciais;

- Métodos mais sofisticados incluem o uso de *Backtracking line search* ou *Exact line search*.

Prós e contras do GDE

✓ Convergência mais rápida, especialmente com grandes bancos de dados ou dados redundantes, p. ex.:

- Imagine que temos dados de treino de tamanho 100.000;

- Mas na verdade são 100 cópias de 1000;

- Ou seja, padrões parecidos, com mesmo efeito;

- Batch será, pelo menos, 100 vezes mais devagar.

✓ A trajetória estocástica permite escapar de um mínimo local;

χ Prova da convergência é probabilística;

χ Muitos métodos de segunda ordem não funcionam;

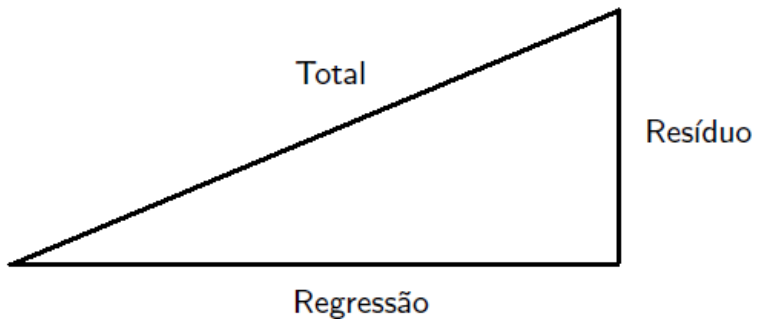
Gradiente boosting

Intuição

- Lembrando de regressão

$$\underbrace{\sum_{i=1}^{n}(y_i-\bar{y})^2}_{SQT}=\underbrace{\sum_{i=1}^{n}(\hat{y}_i-\bar{y})^2}_{SQR}+\underbrace{\sum_{i=1}^{n}(y_i-\hat{y})^2}_{SQE}.$$

- E, geometricamente, temos



- Isso quer dizer que toda variabilidade **não explicada** pela regressão ficará no resíduo (variáveis e funções delas!);

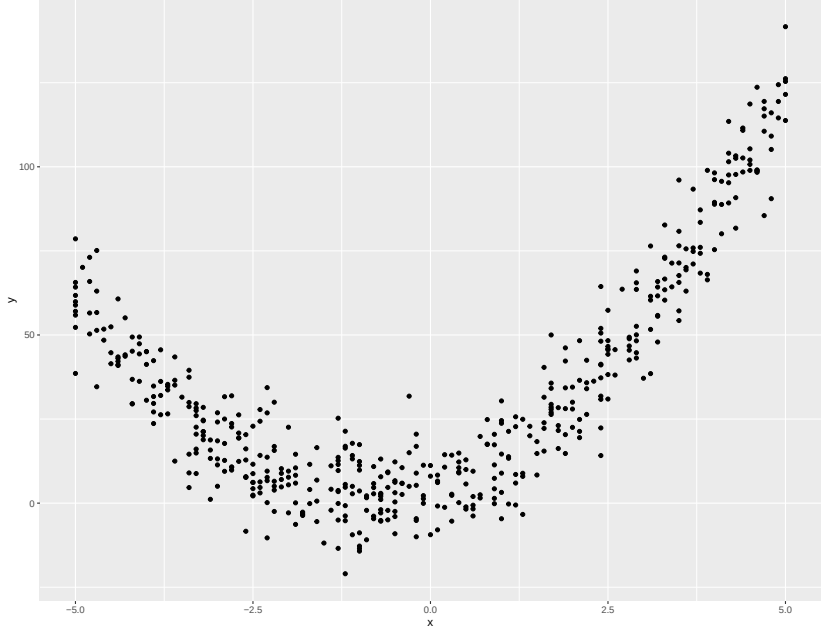
- Vejamos um exemplo.

Intuição

- Simular uma situação na qual a **verdadeira** relação entre X e Y é

$$y = 3,5x^2 + 6x + 5$$

```
x <- sample(seq(from = -5, to = 5, by = 0.1), size = 500, replace = TRUE)
y <- 3.5*x*x + 6*x + 5 + rnorm(500,0,10)
```



Intuição

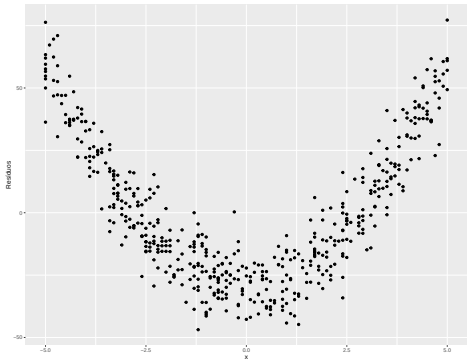
- Agora, vamos ajustar um modelo de regressão linear simples

```
ajuste = lm(y ~ x)
ajuste$coef
```

```
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  33.3499     1.2533   26.61   <2e-16 ***
## x             6.2182     0.4409   14.11   <2e-16 ***
```

- **Pergunta:** Se o termo quadrático não está no modelo, onde ele estará?
- **Resposta:** Nos resíduos.

```
e = ajuste$residuals
qplot(x,e,
      xlab="x",
      ylab="Valores ajustados")
```



Intuição

- Então, considere o seguinte procedimento

$$Y = h(x) + residuo \tag{1}$$

- Se o *residuo* não for um ruído branco (mas algo correlacionado com Y)

$$residuo = g(x) + residuo2 \tag{2}$$

- Combinando (1) e (2)

$$Y = h(x) + g(x) + residuo2$$

- Pode-se dizer que $h(x)$ foi atualizada com uma parte do *residuo*, ou seja

$$h(x)^{(2)} = h(x)^{(1)} + g(x)$$

- Mas, como isto está relacionado com **Gradiente boosting**?

Gradiente boosting e resíduos

- Queremos minimizar

$$J(y_i, h(\boldsymbol{x})) = \frac{1}{2n} \sum_{i=1}^{n} [y_i - h(\boldsymbol{x}_i)]^2$$

- Derivando com relação a $h(\boldsymbol{x}_i)$ temos

$$\frac{\partial J(y_i, h(\boldsymbol{x}))}{\partial h(\boldsymbol{x}_i)} = h(\boldsymbol{x}_i) - y_i.$$

- Podemos interpretar os resíduos como o negativo do gradiente

$$residuos = y_i - h(\boldsymbol{x}_i) = -\frac{\partial J(y_i, h(\boldsymbol{x}))}{\partial h(\boldsymbol{x}_i)}$$

- Então, considerando perda quadrática, concluímos que

resíduo $\Leftrightarrow$ negativo do gradiente
Atualizar $h(\boldsymbol{x}_i)$ com o resíduo $\Leftrightarrow$ Atualizar $h(\boldsymbol{x}_i)$ com o negativo do gradiente

Gradiente boosting

Algoritmo: Escolha um chute inicial, $h(\boldsymbol{x}_i)^{(0)}$, faça:

* Calcule $-\dfrac{\partial J(y_i, h(\boldsymbol{x})^{(k)})}{\partial h(\boldsymbol{x}_i)^{(k)}}$;

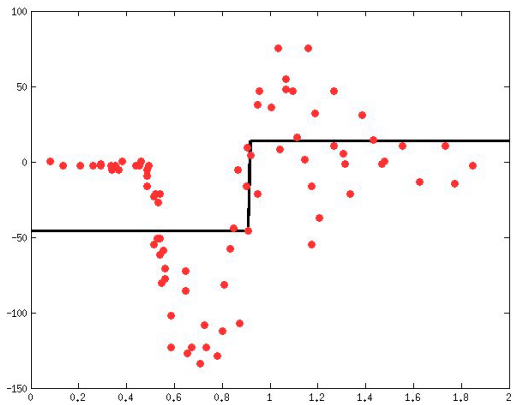
* Ajuste um modelo de regressão $g(\boldsymbol{x}_i)^{(k)}$ baseado no negativo do gradiente;

$$h(\boldsymbol{x}_i)^{(k+1)} = h(\boldsymbol{x}_i)^{(k)} + \rho g(\boldsymbol{x}_i)^{(k)}, \;\; k = 0, 1, \ldots$$

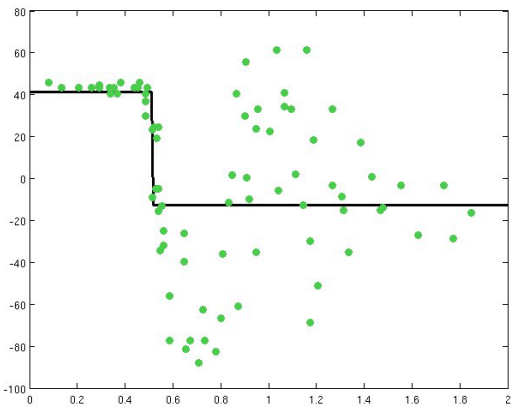
pare quando atingir convergência.

Exemplo:

Começando com um simples preditor



Aprimorando com os resíduos



Gradiente boosting

Algoritmo: Escolha um chute inicial, $h(\boldsymbol{x}_i)^{(0)}$, faça:

* Calcule $-\dfrac{\partial J(y_i, h(\boldsymbol{x})^{(k)})}{\partial h(\boldsymbol{x}_i)^{(k)}}$;

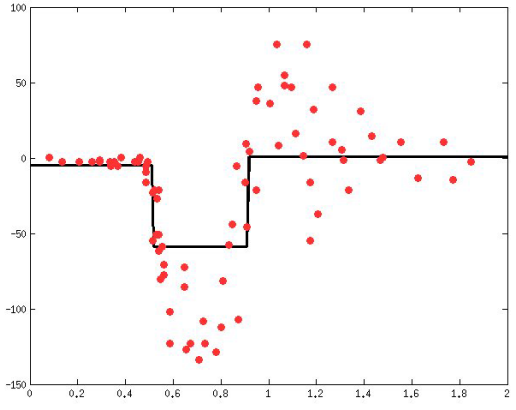
* Ajuste um modelo de regressão $g(\boldsymbol{x}_i)^{(k)}$ baseado no negativo do gradiente;

$$h(\boldsymbol{x}_i)^{(k+1)} = h(\boldsymbol{x}_i)^{(k)} + \rho g(\boldsymbol{x}_i)^{(k)}, \;\; k = 0, 1, \ldots$$

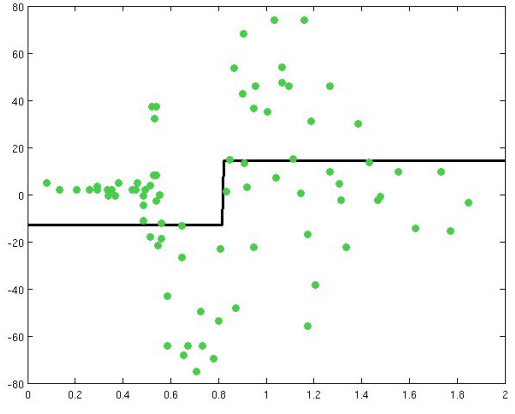
pare quando atingir convergência.

Exemplo:

Combinando, temos um melhor preditor

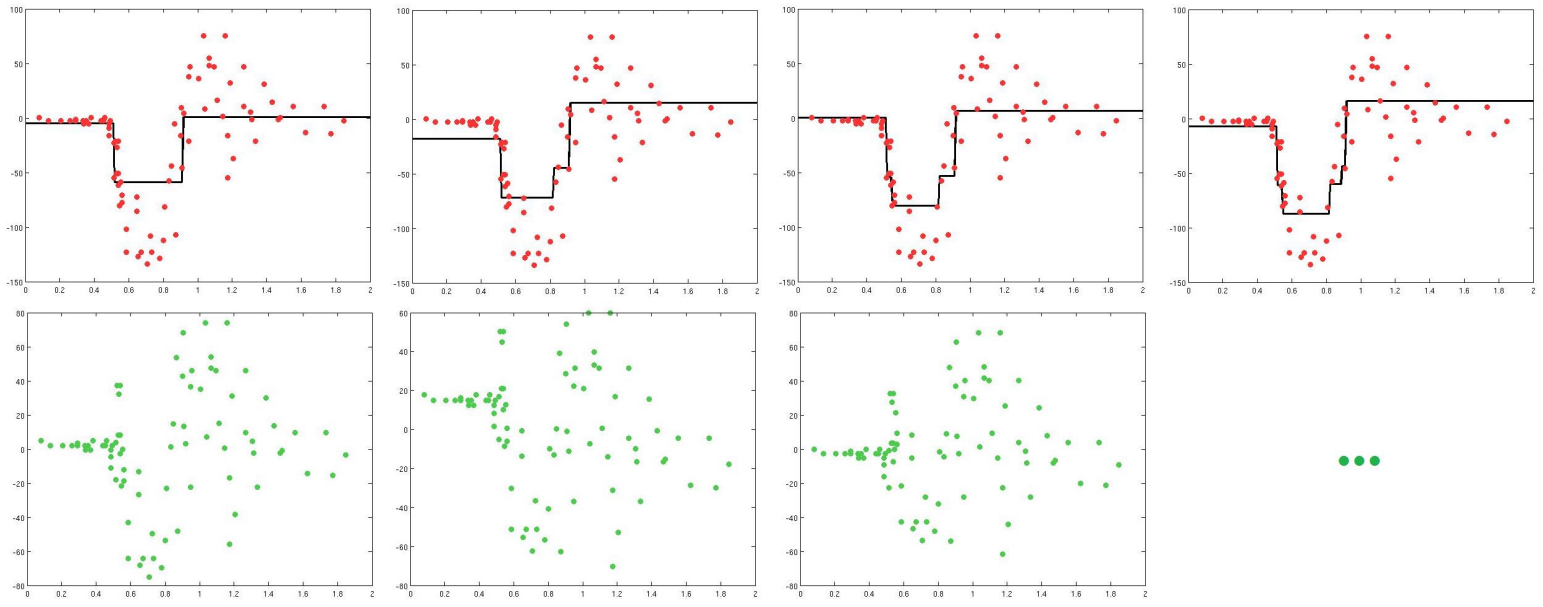


Novamente, aprimorando com os resíduos



Gradiente boosting

- O princípio básico é esse; propor um modelo e aprimorá-lo (ou "ensiná-lo") através da análise dos resíduos;



- Note que podemos considerar outras funções perda e derivar o algoritmo da mesma maneira.

Outras funções perda

- **Soma dos desvios absolutos (SDA)**

$$J(y_i, h(\boldsymbol{x})) = \frac{1}{n}\sum_{i=1}^{n} |y_i - h(\boldsymbol{x}_i)|$$

  - ⋆ O negativo do gradiente fica

$$-\frac{\partial J(y_i, h(\boldsymbol{x}))}{\partial h(\boldsymbol{x}_i)} = sign(y_i - h(\boldsymbol{x}_i)) = \begin{cases} 1, & \text{se } |y_i - h(\boldsymbol{x}_i)| < 0, \\ -1, & \text{se } |y_i - h(\boldsymbol{x}_i)| > 0 \end{cases}$$

- **Huber-M cost**

$$J(y_i, h(\boldsymbol{x})) = \frac{1}{n}\sum_{i=1}^{n} \begin{cases} \frac{1}{2}[y_i - h(\boldsymbol{x}_i)]^2, & \text{para } |y - h(\boldsymbol{x}_i)| \leq \delta, \\ \delta\,|y_i - h(\boldsymbol{x}_i)| - \frac{1}{2}\delta^2, & \text{caso contrário.} \end{cases}$$

  - ⋆ O negativo do gradiente fica

$$-\frac{\partial J(y_i, h(\boldsymbol{x}))}{\partial h(\boldsymbol{x}_i)} = \begin{cases} y_i - h(\boldsymbol{x}_i), & \text{se } |y_i - h(\boldsymbol{x}_i)| \leq \delta, \\ \delta sign(y_i - h(\boldsymbol{x}_i)), & \text{caso contrário.} \end{cases}$$

Conclusão

- O método introduz um novo modelo de regressão em cada iteração, a fim de compensar as deficiências do modelo existente;
- As deficiências são identificadas pelo negativo do gradiente;
- Para qualquer função perda podemos derivar o Gradiente boosting;
- Perda absoluta e Huber são mais robustos a outliers;
- Para detalhes de como escolher o valor de δ, veja
  - Greedy Function Approximation: A Gradient Boosting Machine